INFO0030 : Projet 4

Groupe 33 : Pavlov Aleksandr, Gendebien Alexandre

05.05.2025

1 Architecture générale de code

Notre code est diviser en trois parties : le controller(controller), view(ui) et le model(Leaderboard,game). De manière générale l'utilisateur interagit avec l'ui, qui elle-même interagit avec le controller, qui gère les différentes fonctions d'interaction entre le model et l'ui(nottement les fonction qui demande d'afficher quelque chose à l'écran).

2 Structures de données

Controller :
Structure qui gère l'interaction entre le jeu et l'UI.

Coût en mémoire : faible (principalement des pointeurs et quelques variables).

GameData :
Contient tous les éléments essentiels au jeu : difficulté, taille de la grille, prochaines balles, le score et l'état du jeu (fini ou pas).

Pertinence : rassemble les informations du jeu pour un accès facile.

Coût en mémoire : dépend de la taille du champ de jeu et de la quantité de données stockées (dépend de la difficulté).

Record :
Garde en mémoire le nom du joueur et son score.

Pertinence : utile pour le classement général.

Coût en mémoire : faible, contient relativement peu d'informations.

Leaderboard :
Contient une liste triée des records, permettant l'affichage du classement des joueurs.

Pertinence : permet de conserver l'historique des scores et de les afficher.

Coût en mémoire : dépend du nombre de joueurs enregistrés, mais demande relativement peu de mémoire en général.

Ui :
Décrit l'UI avec divers éléments comme la fenêtre principale, la barre de menu, les boutons du jeu.

Pertinence : permet l'utilisation du système MVC.

Coût en mémoire : principalement des pointeurs vers des éléments GtkWidget, donc utilisation moyenne de la mémoire.

3 Algorithmes particuliers

3.1 Calcul de la carte des distances (calculate_distance_map)

Idée principale : Générer une carte des distances permettant de trouver la distance la plus courte entre ‘start‘ et ‘end‘.

Pour trouver la distance la plus courte entre deux points, on a utilisé un algorithme du type **BFS** ou "Breadth-First Search". L'algorithme BFS explore un graphe ou une grille en visitant tous les voisins d'une case avant de passer au niveau suivant. Il fonctionne par expansion progressive, ce qui garantit qu'il trouve le chemin le plus court dans un environnement où chaque mouvement a le même coût.

3.2 Calcul du chemin optimal (calculate_path)

Idée principale : Reconstruire le chemin optimal à partir de la carte des distances.

Parcours rétrograde à partir de ‘end‘, en suivant les cellules ayant la plus petite distance séparant les deux points, distance donnée par la fonction ‘calculate_distance_map‘. Puis retourne le chemin sous forme de tableau, facilitant le déplacement de la balle.

3.3 Déplacement des balles

Idée principale : Cette fonction gère le déplacement d'une balle d'une position ‘start‘ à une position ‘end‘ sur la grille.

À partir du chemin optimal entre les deux points (‘start‘ et ‘end‘), fourni par la fonction ‘calculate_path‘, la balle est déplacée et l'état du jeu est mis à jour.

Puis met à jour la grille et place les prochaines balles si aucun alignement n'est détecté.

4 Interface Graphique

L'UI est composée d'une barre de menu contenant trois éléments : **Game, Difficulty et Help**.

- Le bouton **Game** contient un sous-menu à trois options : "New Game", "Leaderboard" et "Exit".
- Le deuxième élément de la barre de menu est un bouton qui contient un sous-menu avec les trois options : "Easy", "Medium" et "Hard".
- Le troisième élément de la barre de menu est le bouton "Help", qui affiche une fenêtre popup avec le guide d'utilisation du jeu ainsi que les règles de base.

La partie principale de l'UI est la **grille de jeu**, qui est composée de boutons stockés dans une **GtkVBox**, et les différentes ‘VBox‘ sont stockées dans une **GtkTable**.

C'est sur cette grille que se trouvent les éléments à déplacer (les balles). Entre la barre de menu et la grille :

- Un **label** (à droite de l'écran) affichant le score actuel du joueur.
- Un affichage des **prochaines balles**, qui vont apparaître.

5 Utilisation de GitLab pour la gestion du code

On s'est principalement servi de Git pour partager les modifications que nous apportions au projet lors de son développement. Un point positif que nous avons trouvé en utilisant la plateforme est que nous pouvions documenter nos modifications avant chaque envoi du code. Cela nous a permis d'avancer plus vite, car il est plus facile de se partager le code de cette

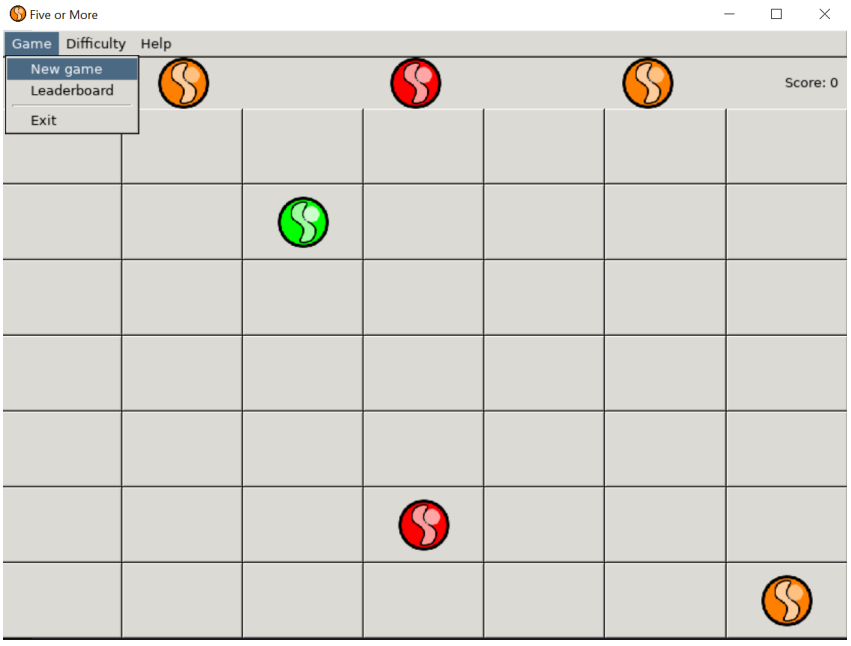

FIGURE 1 – game menu

manière plutôt qu'autrement. De plus, comme nous avons souvent un horaire de travail décalé l'un par rapport à l'autre, la documentation des commits nous permettait de rapidement nous remettre à jour par rapport à ce que l'autre avait fait pendant qu'il avançait sur le projet.

6 Coopération dans le groupe

Comme mentionné précédemment, Git a permis de grandement faciliter le travail et l'échange d'informations sur l'avancement du projet. Mais pour résumer notre coopération : elle était basée sur une liste d'objectifs classés par ordre d'importance. À chaque fois qu'un membre du groupe atteignait un certain quota d'objectifs, il envoyait le résultat sur Git, permettant ainsi au second membre d'avancer sur le projet et de prendre connaissance de ce qui avait été fait.

7 Les améliorations possibles

Comme c'est un jeu relativement basique, il n'y a pas énormément d'améliorations conséquentes que nous pourrions apporter à ce projet, à part :

- Certaines améliorations graphiques.
- Le stockage des 10 meilleurs joueurs avec leurs scores sur un serveur, afin qu'il soit externe au programme et accessible à tous les utilisateurs.

8 Nouveaux éléments appris

Nous avons appris plusieurs choses durant le développement de ce projet, mais les deux plus importantes restent :

- Comment faire une structuration efficace avec des modules indépendants (‘ui‘, ‘game‘, ‘controller‘...).
- Comment utiliser Git pour le suivi des versions et une collaboration plus efficace.

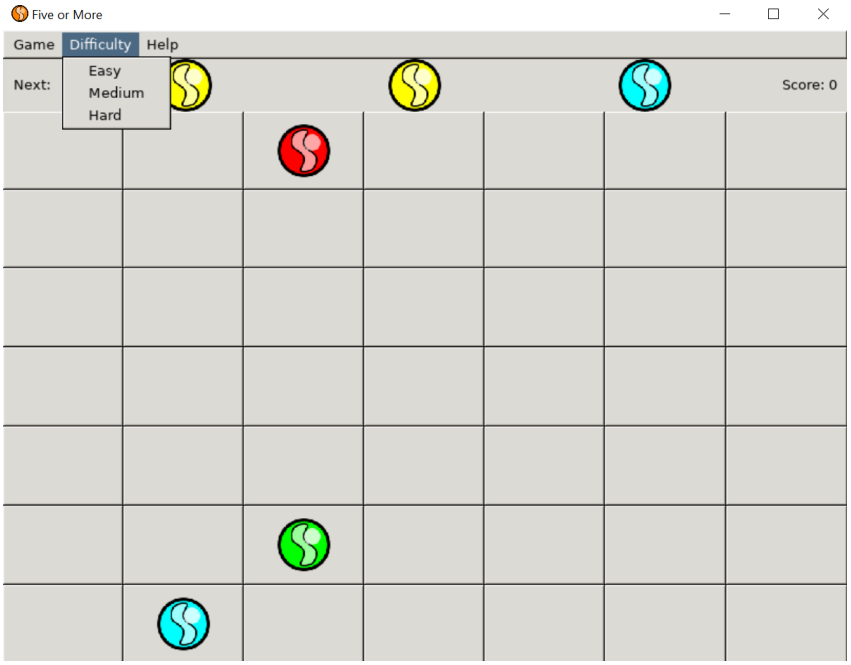

FIGURE 2 – difficulty menu

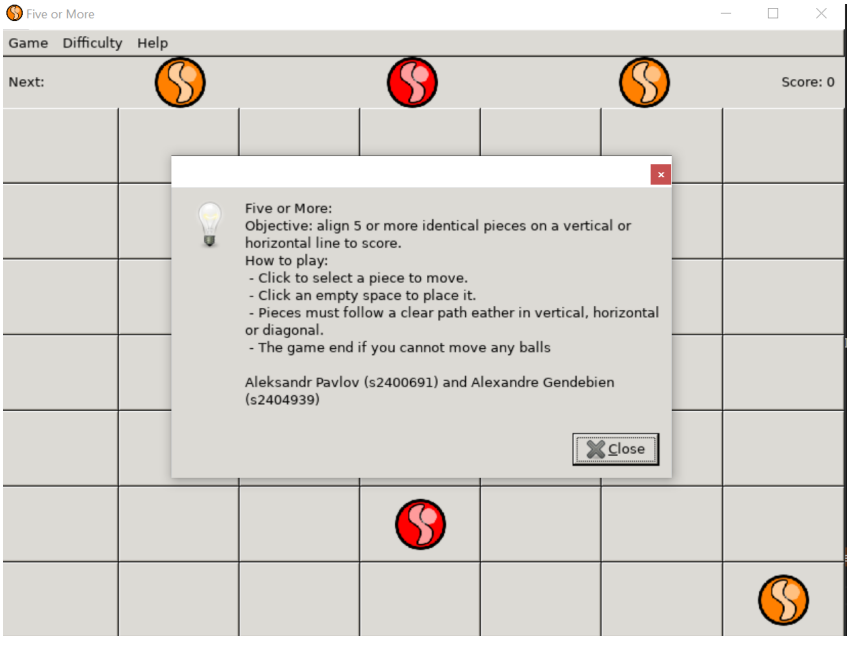

FIGURE 3 – help window